beenen
industrial automation

PLCnext Task Template

C++

Author:
Leon Kristiaan
leonkristiaan@gmail.com

Table of Contents

1 Introduction

Within the PLCnext framework multiple programs can be run real-time simultaneously. These programs are run in Tasks that are in turn managed by the ESM (Execution and Synchronization Manager).
The PLCnext framework allows these programs to be written in one of multiple languages, namely IEC 61131-3, MATLAB Simulink and C++. This document contains information about how the C++ template project works.

This document attempts to provide information about how the template project work in such a manner that it can be used in almost any development environment. There are a few requirements for using this project though, namely:

- **CMake**
- **Make**
- **Phoenix Contact PLCnext SDK**

This template project was built for a Unix environment, though it doesn't matter whether this is *Linux* or *Windows Subsystem for Linux*. The project can be used in different environment but it would require additional modifications.

The GitHub repository for the template project is located at the following address:
https://github.com/Beenen/PLCnext_CPP_Task_Template

2 Filestructure

The following list illustrates the file structure for the template project:

- build
- cmake
  - template
    - *acf.config.in*
    - *esm.config.in*
    - *gds.config.in*
    - *meta.config.in*
    - *libmeta.in*
    - *compmeta.in*
    - *progmeta.in*
    - *opcua.config.in*
  - *CMakeGen.cmake*
  - *ConfMacro.cmake*
  - *ErrorMacro.cmake*
  - *GenMacro.cmake*
- src
  - Programs
    - *TemplateProgram.hpp*
    - *TemplateProgram.cpp*
  - ProgramProviders
    - *TemplateProgramProvider.hpp*
    - *TemplateProgramProvider.cpp*
  - Components
    - *TemplateComponent.hpp*
    - *TemplateComponent.cpp*
  - *TemplateLibrary.hpp*
  - *TemplateLibrary.cpp*
- *CMakeLists.txt*
- *ProjectConfiguration.cmake*
- *CmakePLCnext.sh*

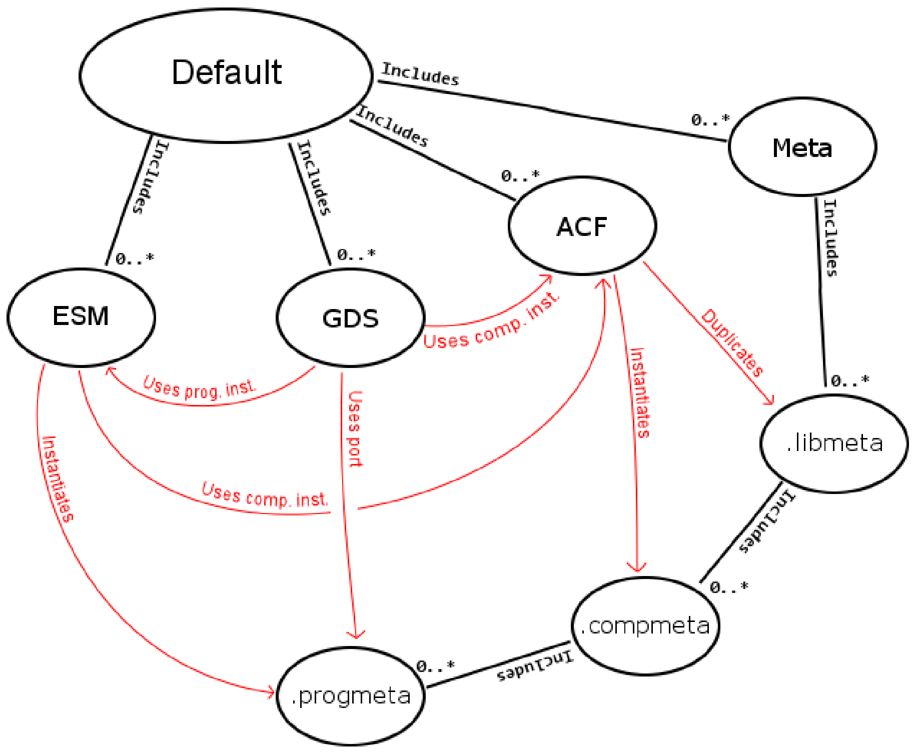
2.1 *build*

The build folder is an empty folder that is used by CMake to house all the build related files. This technique is generally referred to as an “out-of-source” build and serves to separate source code and compilation related files.

2.2 *cmake*

This folder contains all templates and the code that is required to build the configuration files for the PLCnext project using CMake. Only the template files will be discussed in this document as the CMake code has its own internal documentation.

Below is a schematic diagram of the relations between the different configuration files which will be discussed on the following pages:



acf.config.in

These files contain references to the *shared objects* of the libraries in the project and they specify the instances of components within these libraries. The following is an example of this type of file after it has been filled in:

```
<?xml version="1.0" encoding="utf-8"?>
<AcfConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.3"
xmlns="http://www.phoenixcontact.com/schema/acfconfig">
    <Libraries>
        <Library name="TemplateLibrary" binaryPath=
        "$ARP_PROJECTS_DIR$/CPP/Libs/TemplateLibrary/libTemplateLibrary.so"/>
    </Libraries>
    <Components>
        <Component name="TemplateLibrary.TemplateComponentInstance"
        type="TemplateComponent" library="TemplateLibrary">
            <Settings path=""/>
            <Config path=""/>
        </Component>
    </Components>
</AcfConfigurationDocument>
```

Most of the information in these files is similar to what can be found in the .libmeta configuration files. The types of the components are taken from the .compmeta configuration files.

Detailed information about the format of these files can be found here:
https://www.plcnext-community.net/index.php?option=com_content&view=article&id=136&catid=35&Itemid=253&lang=en

esm.config.in

These files contain information about the Tasks that should be created by the ESM and what types of tasks these are. These files define which instances should be made of which programs. They also create links between program instances and tasks that the programs should be part of. The following is an example of this type of file after it has been filled in:

```
<?xml version="1.0" encoding="utf-8"?>
<EsmConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.0"
xmlns="http://www.phoenixcontact.com/schema/esmconfig">
    <Tasks>
        <CyclicTask name="Cyclic1000" stackSize="0" priority="1"
        cycleTime="1000000000" watchdogTime="1000000000"
        executionTimeThreshold="0" />
    </Tasks>
    <EsmTaskRelations>
        <EsmTaskRelation esmName="ESM1" taskName="Cyclic1000" />
    </EsmTaskRelations>
    <Programs>
        <Program name="TemplateProgramInstance" programType="TemplateProgram"
        componentName="TemplateLibrary.TemplateComponentInstance" />
    </Programs>
    <TaskProgramRelations>
        <TaskProgramRelation taskName="Cyclic1000" programName=
        "TemplateLibrary.TemplateComponentInstance/TemplateProgramInstance"
        order="0" />
    </TaskProgramRelations>
</EsmConfigurationDocument>
```

The information in these files about the programs match what can be found in the .progmeta configuration files. The information about the component instances is taken from the ACF configuration files.

Detailed information about the format of these files can be found here:

https://www.plcnext-community.net/index.php?option=com_content&view=article&id=43&catid=35&Itemid=253&lang=en

gds.config.in
These files contain information about the connections that should be made between the in/out-ports of this project and the in/out-ports that are available within the GDS. These connections only have to be made in the GDS configuration for one project. The following is an example of this type of file after it has been filled in:

```xml
<?xml version="1.0" encoding="utf-8"?>
<GdsConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" schemaVersion="1.0"
xmlns="http://www.phoenixcontact.com/schema/gdsconfig">
    <Connectors>
        <Connector startPort=
        "TemplateLibrary.TemplateComponentInstance/TemplateProgramInstance:OP_Byte"
        endPort="Arp.Io.FbIo.PnC/96:Output Byte"/>
    </Connectors>
</GdsConfigurationDocument>
```

The library names and component instance names can be found in the ACF configuration files. The program instance names match those in the ESM configuration files and the port names can be found in the .progmeta configuration files.
Port names for other programs/components can be found in the configurations for the respective projects or in PC Worx Engineer if the program is IEC based.

Detailed information about the format of these files can be found here:
https://www.plcnext-community.net/index.php?option=com_content&view=article&id=68&catid=35&Itemid=253&lang=en

meta.config.in
The meta configuration files only contain references to the locations of the metadata for the project. These files are relatively simple. An example of one that has been filled out is the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<MetaConfigurationDocument>
    <MetaIncludes>
        <MetaInclude path="../../Libs/TemplateLibrary" />
    </MetaIncludes>
</MetaConfigurationDocument>
```

libmeta.in

These metadata configuration files contain information about the libraries and references to metadata files for the components in the libraries. The following is an example of this type of file after it has been filled in:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetaConfigurationDocument xmlns="http://www.phoenixcontact.com/schema/metaconfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.0">
    <Library name="TemplateLibrary" applicationDomain="CPLUSPLUS">
        <File path="libTemplateLibrary.so" checksum="" />
        <ComponentIncludes>
            <Include path="TemplateLibrary_C/TemplateLibrary_C.compmeta"/>
        </ComponentIncludes>
    </Library>
</MetaConfigurationDocument>
```

compmeta.in

These metadata configuration files contain information about the components and references to metadata files for the programs in the components. The following is an example of this type of file after it has been filled in:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetaConfigurationDocument xmlns="http://www.phoenixcontact.com/schema/metaconfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.0">
    <Component name="TemplateComponent">
        <ProgramIncludes>
            <Include path=" TemplateLibrary_P/TemplateLibrary_P.progmeta" />
        </ProgramIncludes>
    </Component>
</MetaConfigurationDocument>
```

progmeta.in

These metadata configuration files contain information about the programs and the ports that are defined within the program. The following is an example of this type of file after it has been filled in:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetaConfigurationDocument schemaVersion="1.0"
xmlns="https://www.phoenixcontact.com/schema/metaconfig">
    <Program name="TemplateProgram">
        <Ports>
            <Port kind="Output" name="OP_Byte" type="uint8" multiplicity="1"/>
        </Ports>
    </Program>
</MetaConfigurationDocument>
```

opcua.config.in
This configuration file contains information about what the OPC UA server can see of the program and how this should be shown. The following is an example of this file after it has been filled in:

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<OpcUAConfigurationDocument xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.phoenixcontact.com/schema/opcuaconfig"
xmlns="http://www.phoenixcontact.com/schema/opcuaconfig">
    <NodeName>TemplateLibrary</NodeName>
    <ServerCertificate>
        <SelfSigned />
    </ServerCertificate>
    <GdsPortsToProvide>
        <All />
    </GdsPortsToProvide>
</OpcUAConfigurationDocument>
```

For as far as I can determine the **NodeName** tag determines under which name this project should be shown by the OPC UA server. Furthermore the **GdsPortsToProvide** determines which ports should be shown by the OPC UA server. I haven't been able to find much information about the valid settings for this beyond **All** and **None**.

2.3 *src*

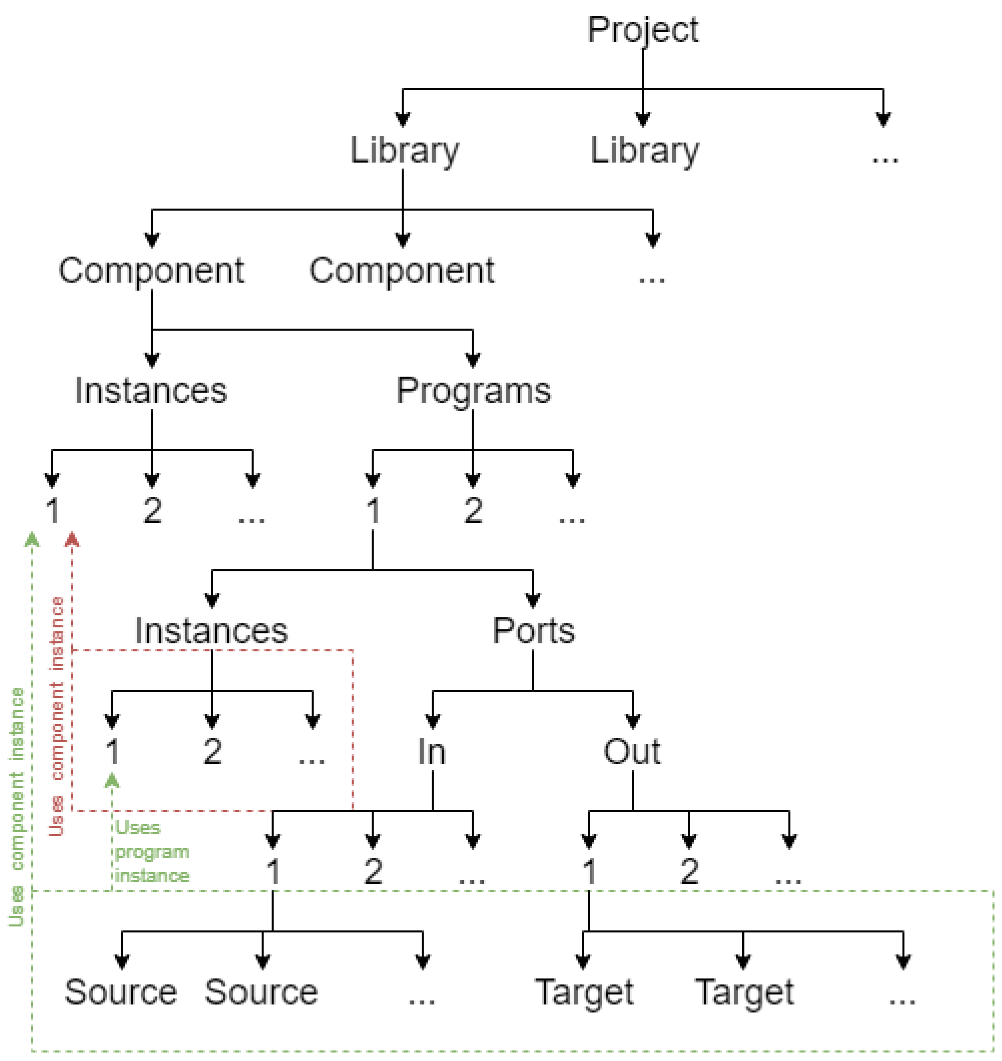
The src folder contains the actual code for the project, this folder contains the code for the libraries, components and programs.
For detailed information about the structure of projects I refer to the example projects of Phoenix Contact and also to their own explanation about this:
https://www.plcnext-community.net/index.php?option=com_content&view=article&id=41&catid=45&Itemid=263&lang=en

3 Configuration

Each project requires its own configuration. To make the creation of this configuration for the developer easier, I have made a number of functions in CMake which build the configuration files. The use of the configuration functions can be done in the following file:

- **ProjectConfiguration.cmake**

Below I have included a schematic diagram of the configuration of a project:



Some explanation of this schematic diagram is required.
A project is the environment in which you work, this doesn't only contain your source code but also configuration files. A project can contain multiple libraries that each have their own purpose. A Library can contain multiple components that each form their own sort of ecosystem in which multiple programs can be made.
The programs realise individual functionalities of a library and are also the place where GDS ports are defined.
To fill each of the layers in this configuration there are functions available in the CMake environment, these will be explained in the following pages.

3.1 *PLCnext*

For standalone projects there are a number of configuration files that need to be adjusted so that the system knows where to find the project. The files that require adjustment are the following:

- `/opt/plcnext/projects/Default/Plc/Esm/Default.esm.config`
- `/opt/plcnext/projects/Default/Plc/Gds/Default.gds.config`
- `/opt/plcnext/projects/Default/Plc/Plm/Plm.acf.config`
- `/opt/plcnext/projects/Default/Plc/Meta/Default.meta.config`

Default.esm.config
This file refers to the location of the esm configuration files of projects. In this file the following lines require changing:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Esm/*.esm.config" />
    <Include path="Globals.esm.config" />
    <Include path="ServiceTask.esm.config" />
</Includes>
```

To these lines a new Include needs to be added, for example:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Esm/*.esm.config" />
```

Default.gds.config
This file refers to the location of the gds configuration files of projects. In this file the following lines require changing:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Gds/*.gds.config" />
</Includes>
```

To these lines a new Include needs to be added, for example:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Gds/*.gds.config" />
```

Plm.acf.config
This file refers to the location of the acf configuration files of projects. In this the following lines require changing:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Plm/*.acf.config" />
</Includes>
```

To these lines a new Include needs to be added, for example:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Plm/*.acf.config" />
```

Default.meta.config
This file refers to the location of the meta configuration files of projects. In this the following lines require changing:

```
<Includes>
    <Include path="$ARP_PROJECTS_DIR$/PCWE/Plc/Meta/*.meta.config" />
</Includes>
```

To these lines a new Include needs to be added, for example:

```
    <Include path="$ARP_PROJECTS_DIR$/CPP/Plc/Meta/*.meta.config" />
```

3.2 *Project*

The project in the context of the configuration refers to the name of the folder within which the project is contained on the PLCnext file system. For IEC application from PC Worx Engineer thi would be “PCWE”, for the template project this is “CPP”.
Also the project requires a path to know where the PLCnext SDK can be found.

`plcnext_library_builder ( exe_path )`

This functions points to the location of the PC Worx Library Builder.

Parameters

`exe_path`

An absolute path pointing to the location of the PC Worx Library Builder.

Example(s)

```
plcnext_library_builder("/opt/EngineeringLibraryBuilder.exe")
```

`plcnext_root_dir ( sdk_path )`

This function points to the location of the PLCnext SDK so CMake knows where to find the header files.

Parameters

`sdk_path`

An absolute path pointing to the root folder of the PLCnext SDK.

Example(s)

```
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
```

`plcnext_project_name ( project_name )`

This function defines the name of the project. This name is used to determine what folder it is in on the PLCnext.

Parameters

`project_name`

The name of the folder in which the project is located on the PLCnext.

Example(s)

```
plcnext_project_name("CPP")
```

`plcnext_add_include ( include_path )`

This function adds a path to the list of include paths. This path points towards external library headers that are required to build the project, for example.

Parameters

`include_path`

A path pointing to the header of an external library for example.

Example(s)

```
plcnext_add_include("/usr/include")
```

3.3 *Libraries*

`plcnext_add_library ( lib_name )`

This function adds a library to the list of libraries that should be built for the project.

Parameters

`lib_name`

A name of a library as found in the source-code.

Example(s)

```
plcnext_add_library("TemplateLib")
```

3.4 *Components*

`plcnext_add_component ( lib_name comp_name )`

This function adds a component to a library.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a component as found in the source-code.

Example(s)

```
plcnext_add_component("TemplateLib" "TemplateComp")
```

`plcnext_add_component_instance ( lib_name comp_name inst_name )`

This function defines an instance of a component.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a component as defined using the *plcnext_add_component* function.

`inst_name`

The name of the instance.

Example(s)

```
plcnext_add_component_instance("TemplateLib" "TemplateComp" "TemplateCompInst")
```

3.5 *Programs*

`plcnext_add_program ( lib_name comp_name prog_name )`

This function adds a program to a component.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a component as defined using the *plcnext_add_component* function.

`prog_name`

The name of a program as found in the source-code.

Example(s)

```
plcnext_add_program("TemplateLib" "TemplateComp" "TemplateProg")
```

`plcnext_add_program_instance ( lib_name comp_name comp_inst prog_name prog_inst )`

This function defines an instance of a program.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a library as defined using the *plcnext_add_component* function.

`comp_inst`

The name of an instance of the *comp_name* as defined using the *plcnext_add_component_instance* function.

`prog_name`

The name of a program as defined using the *plcnext_add_program* function.

`prog_inst`

The name of the instance.

Example(s)

```
plcnext_add_program_instance("TemplateLib" "TemplateComp" "TemplateCompInst"
"TemplateProg" "TemplateProgInst")
```

3.6 *Ports*

`plcnext_add_program_port`
`( lib_name comp_naem prog_name port_name port_type port_mult port_dir )`

This function adds a port to a program. This port matches what can be found in the source-code.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a library as defined using the *plcnext_add_component* function.

`prog_name`

The name of a library as defined using the *plcnext_add_program* function.

`port_name`

The name of the port.

`port_type`

The type of the data in the port. (See Phoenix Contact's list of supported datatypes.)

`port_mult`

The size of the data array in the port.

`port_dir`

The direction of the port, this could in *Output* or *Input*.

Example(s)

```
plcnext_add_program_port("TemplateLib" "TemplateComp" "TemplateProg" "OP_Bit" "bit"
"1" "Output")
```

```
plcnext_add_program_port("TemplateLib" "TemplateComp" "TemplateProg" "OP_Byte"
"uint8" "1" "Output")
```

```
plcnext_add_program_port("TemplateLib" "TemplateComp" "TemplateProg" "IP_Bit" "bit"
"1" "Output")
```

`plcnext_add_program_instance_port_out`
`( lib_name comp_name prog_name prog_inst port_name target )`

This function connects an *Output* port with an *Input* port of another program.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a library as defined using the *plcnext_add_component* function.

`prog_name`

The name of a library as defined using the *plcnext_add_program* function.

`prog_inst`

The name of a program instance as defined using the *plcnext_add_program_instance* function.

`port_name`

The name of an *Output* port as defined using the *plcnext_add_program_port* function.

`target`

A reference to an *Input* port of another program.

Example(s)

```
plcnext_add_program_instance_port_out("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "OP_Bit" "TemplateLib.TemplateCompInst/TemplateProgInst:IP_Bit")
```

```
plcnext_add_program_instance_port_out("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "OP_Byte" "Arp.Plc.Eclr/MainInstance:Input Byte")
```

```
plcnext_add_program_instance_port_out("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "OP_Byte" "Arp.Io.Fbio.Pnc/96:Output Byte")
```

`plcnext_add_program_instance_port_in ( lib_name comp_name prog_name prog_inst port_name source )`

This function connects an *Output* port of another program with an *Input* port.

Parameters

`lib_name`

The name of a library as defined using the *plcnext_add_library* function.

`comp_name`

The name of a library as defined using the *plcnext_add_component* function.

`prog_name`

The name of a library as defined using the *plcnext_add_program* function.

`prog_inst`

The name of a program instance as defined using the *plcnext_add_program_instance* function.

`port_name`

The name of an *Output* port as defined using the *plcnext_add_program_port* function.

`source`

A reference to an *Output* port of another program.

Voorbeeld(en)

```
plcnext_add_program_instance_port_in("TemplateLib" "TemplateComp" "TemplateProg"
"TemplateProgInst" "IP_Bit" "TemplateLib.TemplateCompInst/TemplateProgInst:OP_Bit")
```

3.7 *Tasks*

`plcnext_add_task`
`( task_name task_stack task_prio task_cycle task_watch task_thres )`

This function adds a new cyclic task to the project.

Parameters

`task_name`

The task's name.

`task_stack`

`task_prio`

The priority of the task. This value ranges from 0 to 31 with the lowest value indicating the highest priority.

`task_cycle`

The time between executions of the task measured in nanoseconds.

`task_watch`

The time in nanoseconds until the watchdog triggers if the program is still running.

`task_thres`

Example(s)

```
plcnext_add_task("Cyclic1000" "0" "1" "1000000000" "1000000000" "0")
```

```
plcnext_add_task("Cyclic100" "0" "1" "100000000" "100000000" "0")
```

`plcnext_assign_task ( task_name esm_name )`

This function assigns a task to an ESM instance.

Parameters

`task_name`

The name of a task defined with *plcnext_add_*task.

`esm_name`

The name of an ESM. (ESM1, ESM2, ...)

Example(s)

```
plcnext_assign_task("Cyclic100" "ESM1")
```

`plcnext_assign_program_instance ( lib_name comp_inst prog_inst task_name )`

This function points a program instance to the task which it should be part of.

Parameters

`lib_name`

The name of the library as defined with *plcnext_add_library*.

`comp_inst`

The name of an instance of the *comp_name* component as defined with *plcnext_add_component_instance*.

`prog_inst`

The name of the program instance as defined with *`plcnext_add_program_instance`*.

`task_name`

The name of a task defined with *`plcnext_add_task`*.

Example(s)

```
plcnext_assign_program_instance("TemplateLib" "TemplateCompInst" "TemplateProgInst"
"Cyclic100")
```

3.8 *Examples*

In this sections are a few example configurations using this template.

Below is the template’s configuration file which is set up to contain 1 Library, which has 1 Component and 1 Program. The program has two output ports and those are connected to a Profinet device configured in PC Worx Engineer and a IEC 61131-3 program.
The program instance is also assigned to a task so it can run standalone.

```
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
plcnext_project_name("CPP")

plcnext_add_library ("TemplateLibrary")
plcnext_add_component("TemplateLibrary" "TemplateComponent")
plcnext_add_component_instance("TemplateLibrary" "TemplateComponent"
                               "TemplateComponentInstance")
plcnext_add_program("TemplateLibrary" "TemplateComponent" "TemplateProgram")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "OP_Bit" "bit" "1" "Output")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "OP_Byte" "uint8" "1" "Output")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance")
plcnext_add_program_instance_port_out("TemplateLibrary" "TemplateComponent"
                    "TemplateProgram" "TemplateProgramInstance" "OP_Bit"
                    "Arp.Plc.Eclr/MainInstance:Input Bit")
plcnext_add_program_instance_port_out("TemplateLibrary" "TemplateComponent"
                    "TemplateProgram" "TemplateProgramInstance" "OP_Byte"
                    "Arp.Io.FbIo.PnC/96:Output Byte")

plcnext_add_task("Cyclic1000" "0" "1" "1000000000" "1000000000" "0")
plcnext_assign_task ("Cyclic1000" "ESM1")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance" "Cyclic1000")
```

Below is a configuration which is a variation on the previous one. In this configuration the program only has 1 input port and there are 3 instances of it. Only one program is connected to an IEC 61131-3 program.
This project is also configured to run standalone.

```
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
plcnext_project_name("CPP")

plcnext_add_library ("TemplateLibrary")
plcnext_add_component("TemplateLibrary" "TemplateComponent")
plcnext_add_component_instance("TemplateLibrary" "TemplateComponent"
                               "TemplateComponentInstance")
plcnext_add_program("TemplateLibrary" "TemplateComponent" "TemplateProgram")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "IP_Bit" "bit" "1" "Input")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance1")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance2")
plcnext_add_program_instance("TemplateLibrary" "TemplateComponent"
                             "TemplateComponentInstance" "TemplateProgram"
                             "TemplateProgramInstance3")
plcnext_add_program_instance_port_in("TemplateLibrary" "TemplateComponent"
                    "TemplateProgram" "TemplateProgramInstance3" "IP_Byte"
                    "Arp.Plc.Eclr/:Input")

plcnext_add_task("Cyclic1000" "0" "1" "1000000000" "1000000000" "0")
plcnext_assign_task ("Cyclic1000" "ESM1")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance1" "Cyclic1000")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance2" "Cyclic1000")
plcnext_assign_program_instance("TemplateLibrary" "TemplateComponentInstance"
                                "TemplateProgramInstance3" "Cyclic1000")
```

Below is a minimal configuration that is used for PC Worx Engineer libraries. It specifies the place where the Library Builder can be found. Unlike the others this configuration doesn't specify any instances or tasks, it cannot run standalone.

```
plcnext_library_builder("/opt/EngineeringLibraryBuilder.exe")
plcnext_root_dir("/opt/pxc/2.2.1/sysroots/cortexa9t2hf-neon-pxc-linux-gnueabi")
plcnext_project_name("CPP")

plcnext_add_library ("TemplateLibrary")
plcnext_add_component("TemplateLibrary" "TemplateComponent")
plcnext_add_program("TemplateLibrary" "TemplateComponent" "TemplateProgram")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "IP_Bit" "bit" "1" "Input")
plcnext_add_program_port("TemplateLibrary" "TemplateComponent" "TemplateProgram"
                         "OP_Bit" "bit" "1" "Output")
```

4 Compilation

This template makes use of *CMake*, this is a program which makes compilation configurations for many often used build tools. There is a lot of information online about the use of this program. The template project uses *Make* for its build tool.
It references to the PLCnext SDK from Phoenix Contact as well as (optionally) their Library Builder.

The first step for compiling the project is to generate the Make files using CMake, the template comes with a shell script that does exactly this and which also refers to the PLCnext SDK:

```
./CMakePLCnext.sh
```

Internally this script executes the following command:

```
cmake .. –DCMAKE_INSTALL_PREFIX=../install –DCMAKE_BUILD_TYPE=release
```

This command means execute in the parent folder and place installs in the specified install folder. The configuration for Visual Studio Code comes with tasks configured to execute this command in the **build** folder, this is also the folder in which the configuration files will be placed during generation.

This script refers to the location of the environment setup of the PLCnext SDK:

```
/opt/pxc/2.2.1/environment-setup-cortex9t2hf-neon-pxc-linux-gnueabi
```

This path should be adjusted in case the SDK hasn’t been installed to the standard location.
Once the CMake command has been executed and the Make files have been generated then *make* can be executed in the same folder.

Standalone

For a standalone project for the PLCnext there is a command to prepare the entire project in a separate folder. This allows it to be directly copied to the PLCnext without having to manually collect the built files.

```
make install
```

The folder where the files will be put is the one specified with the **–DCMAKE_INSTALL_PREFIX** option when CMake was run.

PC Worx Engineer Library

For a project which will be imported into PC Worx Engineer as a library there is a different command, this command can only be run when the Library Builder's location has been specified in the configuration.

```
make _PCWE_Build_Library
```

This command prepares a .pcwlx file in the folder in which it is executed, which is normally the **build** folder. This file can be imported into PC Worx Engineer as a library, which will contain the Libraries, Components and Programs as specified in the configuration file of the project.

5 Installation

After a project has been compiled it needs to be installed in the right place. These instructions differ between standalone projects and PC Worx Engineer Libraries.

Standalone

For standalone projects there are some configuration files on the PLCnext that should be adjusted, this step is explained in chapter *3.1: PLCnext*.
For the location of the configuration files of the project itself the name should be kept in mind, this is also the name of the folder in which the standalone project should be placed in the projects folder of the PLCnext.

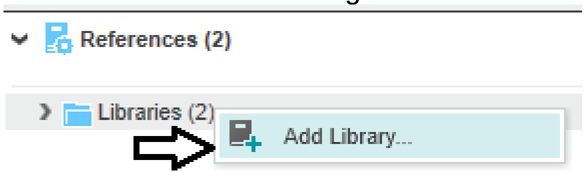
After these configuration steps have been executed the following folder should be created using either the command line or an FTP program:

```
/opt/plcnext/projects/<Naam van project zoals gezet met plcnext_project_name()>
```

As mentioned earlier this is the folder in which the entire install folder can be uploaded.

PC Worx Engineer Library

For PC Worx Engineer Libraries it is pretty simple, in the bottom right there is an area in which “References” can be found. When right-clicking on the “Libraries” item in that area a drop-down will be shown with the following:



After clicking this a dialog is shown with which the .pcwlx file can be located and loaded. This makes the programs from the libraries available for configuration in PC Worx Engineer.